ARCADA

Feature-based Image Interpretation for Lightweight Computer Vision

Erik Granholm

Erik Granholm

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	5205
Författare:	Erik Granholm
Arbetets namn:	Feature based Image Interpretation for Lightweight Computer Vision
Handledare (Arcada):	Jonny Karlsson
Uppdragsgivare:	

Sammandrag:

Datorseende är ett område inom datavetenskap som innebär att extrahera underliggande information ur bilddata, med avsikt att kunna agera utgående från bildens innehåll. Människan kan lätt tolka färg- och rymdinformation, särskilja för- och bakgrund ifrån varandra samt följa och klassificera objekt och personer från video. Även med ständigt ökande beräkningskapacitet har datorer svårigheter att nå nivån av ett två år gammalt barn. Detta examensarbete utvärderar bildbehandlings algoritmer, samt algoritmer för upptäckning, beskrivning samt jämförelse av såkallade bildkännetecken. Dylika algoritmer har även implementerats i ett funktionsbibliotek för självständig och resurssnål datoseende, som också beskrivs i detta examensarbete.

Nyckelord:	Computer vision, scale-Invariant feature transform, image features
Sidantal:	49
Språk:	Engelska
Datum för godkännande:	01/06/15

Erik Granholm

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	5205
Author:	Erik Granholm
Title:	Feature based Image Interpretation for Lightweight Computer Vision
Supervisor (Arcada):	Jonny Karlsson
Commissioned by:	

Abstract:

Computer vision is a field in computer science which aims to process, analyse and understand images in order to extract underlying information from them and to act on the information. Humans are able to extract information from visual stimuli with apparent ease. We can interpret distances and color information, distinguish fore- and background as well as identify people and classify objects into groups. We can follow moving objects in a consecutive sequence of images and accurately estimate spatial information from a video. Even with ever increasing processing power, computers have yet to reach the level of a two-year old child in many of the applications. This thesis evaluates image processing, feature detecting, description and matching algorithms in computer vision with emphasis on scale invariant feature transform. A new independent and light weight computer vision library has also been developed implementing some of the described algorithms. An overview of this library is also provided in this thesis.

Keywords:	Computer vision, scale-Invariant feature transform, image features
Number of pages:	49
Language:	English
Date of acceptance:	01/06/15

CONTENTS

Figures

1 INTRODUCTION

Computer vision is a field in computer science which aims to process, analyse and understand images in order to extract underlying information from them and to act on the information. Humans are able to extract information from visual stimuli with apparent ease. We can interpret distances and color information, distinguish fore- and background as well as identify people and classify objects into groups. We can follow moving objects in a consecutive sequence of images and accurately estimate spatial information from a video (Szeliski 2011).

Today computer vision has applications in a wide assortment of industries (Mettler-Toledo, 2013) and services, including medicine, engineering and surveillance/security. In engineering industries, computer vision applications are used to inspect e.g. steel castings or auto-mobile and aircraft parts for defects. This process is called machine inspection or automated optical inspection (AOI). There are advantages to this approach compared to having persons manually inspect the object. The object under inspection may be under specialized illumination, but no direct contact with the object is necessary (Szeliski, 2006). The automated inspection can also be done in several stages of production. In mass production this approach can be used, instead of for example random sampling, in order to improve quality-statistics and reduce risk of packaging a defected product.

In medicine, segmentation algorithms such as active contours, also called snakes, are used as a tool for analysing and visualizing medical images such as brain scans and magnetic resonance images. In medical surgery, computer vision algorithms can be used to create models of the patients´ structural and functional anatomy and vascular structure. Diagnoses of diseases can also be done by interpreting images, for example automatic diagnosis of tuberculosis using a mobile microscope. (Chang, 2012).

Intrusion detection, highway traffic analysis and other security applications such as monitoring of public areas like swimming pools help people in decision-making and reacting to risky situations. All these brilliant applications make it seem as if the engineers and scientists have computer vision covered. In reality, vision is an extremely hard sense to emulate. Modern applications have yet to surpass the level of an infant child in many tasks, such as classification of elements in an image. The challenge in emulating vision is in its

inverse nature. Where computers usually calculate any results from measurements, computer vision attempts to reconstruct the underlying information from the measurements. This is called an inverse problem. Usually the measurements (images) contain "insufficient information for us to fully specify the solution" (Szeliski, 2006). Even though improvements are made all the time, computer vision algorithms are error prone. People unfamiliar with the subject usually underestimate the difficulty of these problems, or as neatly put by XKCD in figure 1:

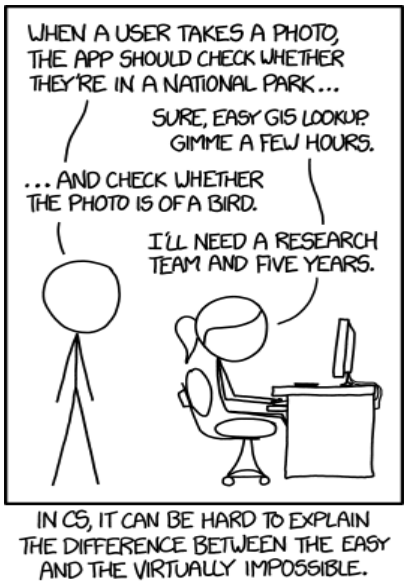

Figure 1: Computer vision themed comic by XKCD (Munroe)

A lot of research is done on the subject computer vision, and it is a broad field indeed. With the increasing processing power of of microchips and processors and the shrinking size of cameras, it is becoming plausible to run computer vision applications on hand-held and other portable platforms. This thesis will concentrate on feature detecting, describing and matching and attempt to pick a suitable combination of image processing algorithms for a lightweight general purpose computer vision library. It will also attempt to optimize it for minimal usage

of computer resources. Other similar implementations such as the popular OpenCV (OpenCV, 2015), OpenSIFT (Hess, 2010) and VLFeat (VLFeat, 2015) contain a variety of computer vision algorithms.

OpenCV is probably the most widely used library today. It is developed by Intel in C++, but has interfaces to a wide variety of programming languages, and a huge variety of computer vision algorithms. OpenSIFT is an implementation of the Scale-Invariant feature transform, and is developed in C using the openCV function library. VLFeat is a collection of cross-plattform computer vision algorithms written in C and MATLAB.

This thesis presents an implementation of the scale-Invariant feature transform which is influenced by the OpenSIFT implementation. A new function library for image processing and computer vision has also been implemented to achieve independency from existing libraries. Since existing libraries may have dependencies and requirements of their own, they are not suitable for the long term pruposes of the presented library. The long term purposes include becoming an independent all around image processing and computer vision library. The scope of the practical work is limited to the algorithms themselves. No fully working end-product will be produced, as a complete software is irrelevant to the purpose of this thesis.

The rest of this thesis is structured as follows; Chapter 2 discusses the background and theory realted to the image processing and feature detector/descriptor implementations presented later in this thesis. In Chapter 3 some image processing implementations and theory central to computer vision are presented including the implemented difference of gaussian feature detector and the SIFT (scale invariant feature transform) descriptor. An overview of the implemented library as well as a performance evaluation is presented in Chapter 4, followed by Conclusions and discussion in Chapter 5.

2 BACKGROUND AND THEORY

2.1 Human Trichromacy

Humans have trichromatic vision, meaning we have three different photoreceptor cells for interpreting color information from visible light. These cells are located in the retina and are

called cone cells. They operate in bright light and can be divided into three categories depending on the wavelengths they are responsive to. L-cones are responsive to long wavelengths which corresponds to red color, M-cones are responsive to medium length wavelength, which corresponds to green color and S-cones respond to short wavelengths, corresponding to blue color. Since humans have more of the M and L types of cones, we are more sensitive to yellowish green light than red or blue, since the M and L cones are both highly responsive in bright light (Schubert, 2006). There have been confirmed cases of slight tetra chromatic vision in humans, mostly in females (Hays, 2013a). Apart from cones, humans also have rod-cells, which are highly sensitive to light and are responsible for gray scale vision and night vision. See figure 2 for the photometric standard that describes the human vision, called the human eye sensitivity function. (Hays, 2013a)

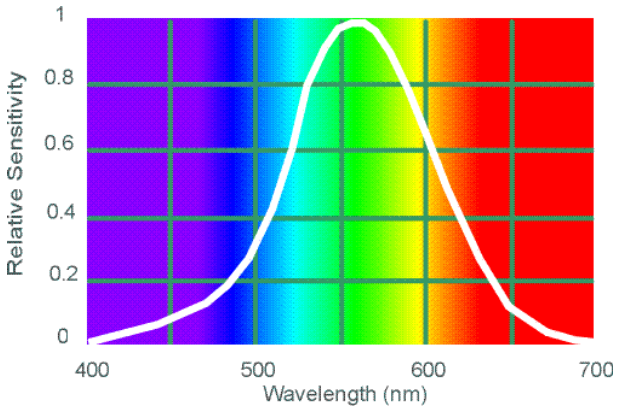

Figure 2: The human eye sensitivity function (Hays, 2013a)

2.2 Image Formation

The life cycle of a photon in the context of imaging is roughly as follows. Photons emitted from an illumination source, such as the sun or a light bulb, strike the surface of an object. At this point a part of the photons are absorbed by the surface of the object, and a part is reflected. The reflected photons are captured by an imaging system and projected (captured)

on a surface. In conventional cameras the light is captured on a light-sensitive film. In digital cameras, the film is replaced with a sensor array. The sensor array consists of a surface of light-sensitive diodes that convert photons into electrons. Light that hits the sensor array is sampled and quantized, which turns the analogue image into a digital image (Hays, 2013a).

Color information is typically captured with a color filter mosaic called a Bayer filter, or Bayer grid, named after its inventor Bryce Bayer. It consists of a pattern of red, green and blue color filters, such that the green filters make up 50 percent of the area, and the remaining red and blue 25 percent each. The motivation behind the distribution of colors in the mosaic is to mimic the human vision, which as discussed above, is more sensitive to green light as opposed to red and blue. Post sampling and quantization, the output of the Bayer filter is called the Bayer input pattern. Since each filter in the pattern can sense only one of the three colors the image is incomplete and the missing color information needs to be interpolated from the surrounding color information with a demosaicing algorithm, several such algorithms exists, and the quality of the resulting image depends on the chosen algorithm.

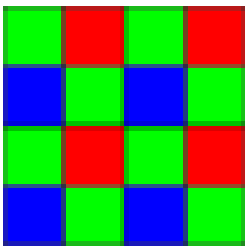

Figure 3: Bayer input pattern

2.2.1 Color Spaces

In data, color can be modelled in several different ways using a representation referred to as the color model, or color space. The color space is a mathematical representation of a color using combinations of usually two to four values. They can be divided into additive and subtractive models, both of which make use of three primary colors, and sometimes an extra

alpha channel is added to provide support for opacity. The primary colors for additive models are red, green and blue, and the three primary colors for subtractive models are cyan, magenta and yellow. Additive colors generate colors by adding them together, much like mixing paint of different colors to create new ones, while subtractive models are subtracted from each other, much like in nature wavelengths are absorbed by illuminated surfaces. Typically used additive representations are the HSV (Hue, Saturation, Value) and HSL (Hue, Saturation, Lightness) models. A popular subtractive model is the CMYK (Cyan, Magenta, Yellow, Key).

HSL/HSV:

HSL and HSV are represented as a three dimensional color-wheel in the shape of a cylinder, where the base is a gradient of the three primary and secondary colors. By picking an angle one picks a color. For example red can be found at an angle of 0 or 360 degrees, green at 120 degrees, and blue at 240 degrees. The length of a vector starting from the centre of the wheel indicates the saturation of the color, and the height of the cylinder either lightness or value depending on the representation, see Figure 3. For detailed pseudo code conversion between RGB and HSL/HSV, refer to (Agoston 2005).

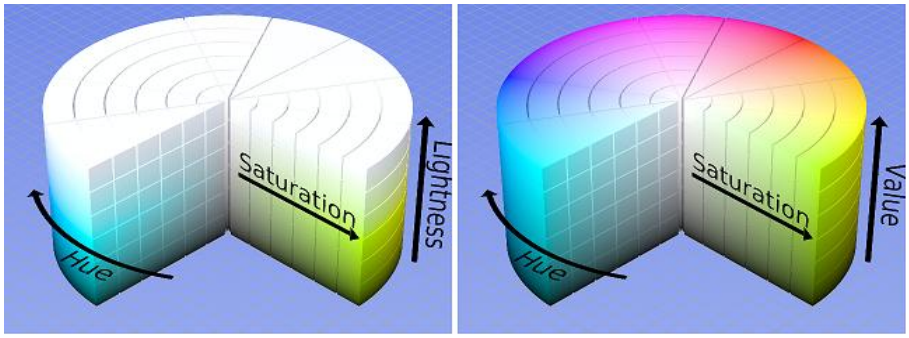

Figure 4: The HSL and HSV color models visualized (Unknown, Image2)

CMYK:

CMYK is shorthand for the three primary colors (plus black) that are used in subtractive color models; cyan, magenta, yellow and Key (Black). The black channel is added mainly for practical reasons, since the CMYK color space is mostly used in printer devices, and achieving pure black using the three remaining channels is both uneconomic and more error prone. The idea of subtractive color spaces is that they subtract certain wavelengths from white light, producing new colors. Hence, the CMYK model is appropriate for printing on white paper.

2.3 Notation

We will consider images as three dimensional functions with two spatial parameters, the values of which correspond to the sampled pixel values. In theory the image functions can be considered of infinite resolution, in practice sub pixel values have to be interpolated from the neighbouring sampled values. In grey scale images a high image function value corresponds to a light color, with full white at maximum pixel value, and full black at zero. In color images a function value corresponds to the corresponding color channels intensity, such that for RGB (Red Green Blue) images three such values exist for each sampled pixel. Input images are are denoted:

$$I(x, y)$$

Where x and y are the spatial parameters, such that *x=0, y=0* is the top-left sample, and the x-axis is increasing to the right and the y-axis is increasing downwards.

2.4 Feature Engineering

Features can be considered as regions in images which hold a certain kind of valuable information. The human vision is naturally drawn towards features with for example strong contrast, clear edges and distinct shapes. From an image containing ten black circles and one red triangle, a human tends to pay attention to the red triangle. In this case the triangle is interesting to us since it has pointy features (corners) and distinguishable color contrast with

regards to the other shapes in the image. The life cycle of a feature, in terms of feature based recognition consists typically of three stages. The features are first detected in a stage called feature detection, after which they are described in a way such that they can efficiently and reliably be distinguished from another. Lastly they are matched against features from training images in order to determine whether the features in question match or not (Hays, 2013a) (Szeliski 2011).

2.4.1 The Feature Detector

In computer vision features are detected by a process usually referred to as 'feature detection'. A feature detection algorithm scans an entire image at pixel level and attempts to determine whether a region around location *I(x, y)* contains a feature or not. What constitutes a feature is defined in the logic of the detector. Typically feature detectors are interested in features such as interest points, corners, edges or blobs/regions. Image features are used in computer vision applications such as image alignment, Object recognition and motion tracking. Thanks to the local nature of image features they are well suited for recognition applications even in cases where the object being detected is partially occluded. Another significant advantage is the quantity of features, which can be in the thousands. For small objects with less probability for false matches as few as three features can be sufficient for reliable correct matching (Szeliski, 2011).

Edges:

The definition of an edge is a clear boundary between image regions, edges can be of any shape. Edges lead to sudden changes in the images intensity function, and can also be detected by finding the extrema of an images first derivative.

Corners and interest points:

A corner is any area with a pointy-like feature, typically detected by analysing the local curvature in the image gradients. Interest points are points in images with high contrast (black dot against white background, or vice versa), they are typically detected in a similar fashion as corners.

Blobs:

Blobs are regions of images that differ in properties compared to the regions surrounding them. They are usually detected around an interest point.

The quality of a feature, for the purpose of reliable matching, can be measured by its uniqueness, meaning how distinct it is when compared to any other feature. Consider the four features in Figure 4. Some features can be matched to several locations, while other can only be matched against a single location in the image. Image patches with low to no texture or low contrast are hard to localize, and can therefore for the purpose of reliable feature matching be considered of poor quality.

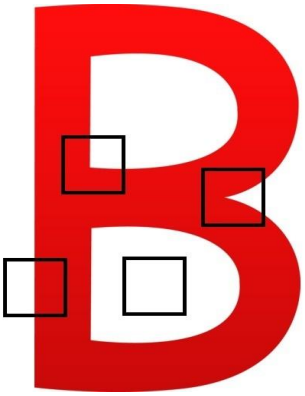

Figure 5: Examples of features of both good and poor quality.

The Quality of a feature detector can be measured by its repeatability. This method was first proposed by Schmid, Mohr and Bauckhage (Schmid, 2000). They defined repeatability as the frequency with which a detected feature keypoint could be detected in a set of images that had undergone adding of noise and various transformations. These transformations included changes in scale, viewpoint, rotation and illumination. A keypoint was deemed repeated if it was re-detected in the transformed images within ϵ pixels in the corresponding original

image. Some popular detectors are the Harris corner detector, difference of Gaussian/SIFT detector (Lowe, 2014) and Canny detector (Hays, 2013c).

The function library developed within the scope of this thesis implements the difference of Gaussian (DoG) feature detector, as it has many advantageous attributes. Both the library and the DoG detector are described in Chapter 3.The Feature Descriptor

The purpose of the feature descriptor is to reduce the image regions dimensionality as much as possible, while still being able to describe the feature region in a way that is invariant to as many factors as possible. Some of these factors can be geometric transformations, changes in luminance and viewpoint or image noise. According to descriptor comparisons made by Mikolajczyk and Schmid the best performing descriptor was the GLOH (Gradient Location and Orientation Histogram) descriptor, SIFT (Scale-Invariant Feature Transform) was a close second. They both make use of local image gradients collected into several orientation histograms. (Mikolajczyk, 2000)

The SIFT descriptor, described in detail in Chapter 3, was chosen for the function library since it has good tolerance against many problematic factors.

2.4.2 Feature Matching

After the descriptors have been extracted from local keypoints, the next step is to reliably match them to corresponding locations in other images. The simplest matching logic is to use the nearest neighbour approach, calculating the euclidean distance between two descriptors, and record the keypoint as matched if the distance is within given threshold value. Determining a proper threshold value is critical, since setting it too low will increase false negatives, and setting it too high will increase false positives. In order to keep true positives, but also discard false positives it is possible to compare the ratio between the best match and the second best match, as suggested by Lowe in his paper *distinctive features from scale invariant keypoints* (Lowe, 2004). His approach discarded matches where the distance ratio was greater than 0.8. His results showed that this eliminated 90% of false positives, and only discarded 5% true positives.

In cases where the amount of stored descriptors is large some preprocessing may be necessary. By creating an indexing structure (like a KD-tree or hash table) for storing keypoints it is possible to reduce the amount of matching candidates in order to increase matching speed. (Lowe, 2004)

2.5 The Gaussian Function

The Gaussian kernel is widely used within image processing algorithms, and is of special interest for the implementations presented in this thesis. It is named after its German inventor, mathematician Carl Friedrich Gauss, and is defined:

in 1 dimension:

$$G(x;\sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}$$

in 2 dimensions:

$$G(x,y;\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

in N dimensions:

$$G(\vec{x};\sigma) = \frac{1}{(\sqrt{2\pi\sigma})^n} e^{-\frac{|x|^2}{2\sigma^2}}$$

The sigma-parameter, also called the inner-scale (or scale for short) determines the width of the Gaussian function. The sigma parameter is more than a simple spatial parameter, it is per convention denoted here separated by a semicolon, to distinguish it from the other spatial parameters.

The Gaussian function is a normalized function, meaning its integral value from negative to positive infinity is always unity, independent of the supplied scale parameter. This leads to a significant drop in the functions magnitude as the scale (width) is increased. Consider plotted 2 dimensional Gaussian curves with descending sigma values in Figure 6.

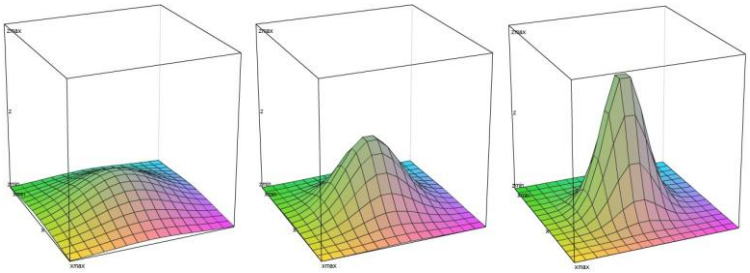

Figure 6: 2 dimensional Gaussian curves with sigma values descending to the right.

The Gaussian is a low-pass filter, meaning it will pass low frequencies and attenuate higher frequencies. When applied to an image, the 2 dimensional Gaussian kernel has an average gray level invariant blurring effect, meaning the gray level of the image is not affected after the filter is applied. For the implementations presented, the Gaussian kernel is assumed to be isotropic meaning, equal sigma for each spatial dimension, symmetric around Z-axis. The Gaussian filter is also separable, separable filters are discussed in Chapter 3.

3 IMPLEMENTATIONS

This chapter presents some of the library's functionality. These techniques and algorithms were selected to be part of the library since they are popular and useful computer vision basic operations. The implementations of the PPM (Portable Pixel Map) image formats, image filtering, and feature implementations, among others, are discussed in detail in this Chapter.

3.1 The PPM and floating point image formats

The implementations discussed in this thesis make use of either PPM or floating point images. PPM is defined by the Netpbm project (Netpbm, 2013). The PPM file format was chosen as

the default input format for the implementations presented in this thesis, as it is trivially implemented.

The format has significant disadvantages; it is highly inefficient, and supports only simple color information. The actual pixel data can be one of two encodings; ASCII or raw binary, and one of three color models; RGB, gray scale or bitmap. The three color models are called portable pixmap, portable gray map and portable bitmap, and are sometimes collectively referred to as portable anymaps. It was chosen that the implementations presented in this thesis would not support the ASCII format, since it requires unreasonable amounts of memory. A PPM file can be divided into its ASCII header and data raster. A PPM header consists of a PPM magic number, defining the color model and data-type of the file, image dimensions and pixel maximum value, all delimited by whitespace. The maximum pixel value is expected to be within 1 and 65536, and defines the color depth. The header is followed by a whitespace and the image pixel data. Implemented PPM format structure implemented in the function library can be seen in Figure 7. For convenience purposes, an additional *data length* field was added. Its value is the amount of bytes needed to allocate *width * height* pixels for a certain color model.

```
/**
 * represents a PPM header
 */
struct PPM_HEADER{
    unsigned int type;
    unsigned int width;
    unsigned int height;
    unsigned int data_length;
    unsigned int maxval;
};

/**
 * represents PPM image data
 */
struct PPM_IMAGE{
    struct PPM_HEADER header;
    uint8_t *data;
};
```

Figure 7: The PPM file structure

The image data is arranged left to right in a single dimensioned array of 8-bit unsigned integers, see Figure 8. This may seem counter-intuitive, as images typically are represented as

two dimensional structures. This representation was chosen for simplicity reasons, as the C programming language does not directly support multidimensional arrays, and 'pointer-to-pointer' type arrays reside adjacent in memory anyway. A single color channel needs 8 bits/pixel, with the exception of the bitmap format, which only needs a single bit per pixel. Color channels are arranged from left to right *red, green, blue*.

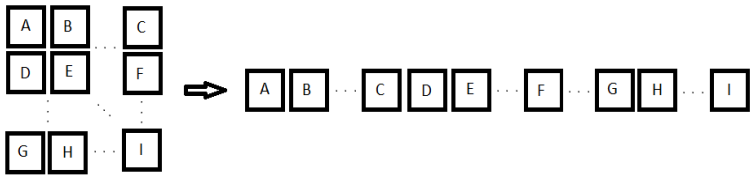

Figure 8: Elements in 2 dimensions rearranged into 1 dimension. Similar approach as in Mylers book (Myler)

A pixel value can be sampled with

$$I(x, y) = I(((y * w * c) + (x * c)) + n)$$

where *w* is the width of the image, *c* the count of color channels, *n* the color channel being sampled and *x*, *y* the spatial parameters. For the implementations that require it, a PPM file can be converted into a floating point image, , which is implemented in a very similar fashion.

```
/*
 * Represents a floating point image header,
 * where pixel values are expected to be
 * within 0 <= p <= 1
 */
struct FLOAT_HEADER {
    int width;
    int height;
    int channels;
    int data_length;
};


/*
 * represents a floating point image
 */
struct FLOAT_IMAGE {
    struct FLOAT_HEADER header;
    double *data;
};
```

Figure 9: The floating point image structure

I

3.2 Image Processing

This Section presents a couple of image pre-processing implementations, as pre-processing is central to many computer vision algorithms. Implementations of convolution filtering and skin color classification are presented in this chapter, since they are popular and useful for many modern applications such as face- and feature detection, including the SIFT described in Chapter 3.3.

3.2.1 Convolution Filters

Convolution filtering is very central to many computer vision algorithms, and can be a very time-expensive operation, especially if the target image and filter kernel are large. Convolution filtering is a convolution operation between the chosen filter and the image neighbourhood, the resulting value is stored in the target pixel. Pseudo code for the convolution filtering can be seen in Figure 10.

```
FOR 0 TO image height
FOR 0 TO image width

        SET pixel sum to 0

        FOR 0 TO filter height
        FOR 0 TO filter width

                COMPUTE image coordinates, consider image boundaries
                COMPUTE pixel value from image coordinates
                MULTIPLY filter value with pixel value, SET to temp value
                ADD temp value to pixel sum

        ENDFOR
        ENDFOR

        ADD filter bias to pixel sum
        MULTIPLY pixel sum with filter factor, SET to output value
        STORE output value in output image

ENDFOR
ENDFOR
```

Figure 10: Convolution filtering can be implemented with 4 levels of nested for loops

In practice, this means weighing the image neighbourhood with the filters values and the sum of the resulting neighbourhood becomes the value of the target pixel. The process of convolution, such that *K* is the width of the kernel, requires *K^2* operations (denoted *O*) per pixel. The total amount of operations can therefore be calculated *W*H*O* where *W* and *H* are the width and height of the image, which implies linear time complexity. If the Sum of the filter values does not equal zero, the output image will become darker for filter sums below zero, and lighter for filter sums larger than zero. To avoid unwanted changes in the output images colors, a filter factor can be calculated by dividing 1 with the sum of the filter values, the final pixel value is multiplied with the corresponding filters factor. However, would such an effect be desirable a filter bias constant can be added to the target pixels value prior to multiplying it with the factor to achieve appropriate lightening or darkening of the output image. Certain filters can be separated into two filters, such that the product of the two separated filters becomes the original filter. This allows for faster processing, and is generally a good idea for large filter kernels. These types of filters are called separable filters, and are yet to be implemented into this library.

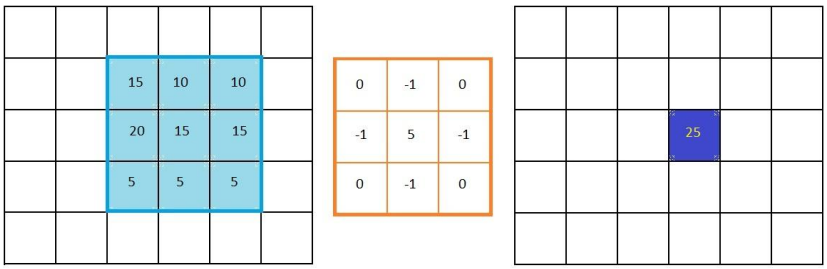

Figure 11: Target image neighborhood, filter region and output pixel visualized

Apart from blurring or sharpening, filters are used for a variety of operations, such as corner and edge detection, calculation of image gradients or distance transforms. Some useful filters include the low pass Gaussian filter with a blurring effect and the Sobel operator, used for calculating estimated image derivatives. As an example the implemented Sobel operator, also known as the Sobel filter, is presented since it is popular in edge detection algorithms, and a similar approach for calculating image gradients and magnitudes is used in the SIFT presented in Chapter 3. The Sobel operator is a linear filter which consists of two separate filters. Both Sobel filters are separable and can be seen in Figure 12. The purpose of the filters is to calculate approximations of horizontal (changes in x-axis) and vertical (changes in y-axis) image derivatives. The input image is filtered with both Sobel filters and the resulting two images, *Sx* and *Sy*, contain the derivative approximation for their respective dimensions.

$$Sx = I * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad Sy = I * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 12: The Sobel filters in x-axis (left) and y-axis (right)

From the two derivative approximations, the images gradient magnitudes and orientations can be calculated with:

$$\Theta = atan2(Sy, Sx)$$
$$m = \sqrt{Sx^2 + Sy^2}$$

Examples of image gradient orientations and gradient magnitudes calculated with the Sobel operator can be seen in Figure 13.

3.2.2 Skin Color Classifier

A skin color detection sweep evaluates the pixel color values to determine whether a pixel is of skin color or not. They are useful as a pre-processing stage in for example face detection algorithms. This kind of operation is not a neighbourhood operation, and requires only the inspection of the target pixel during computation. It is faster to compute than large convolution filters. The skin color detector implemented in this library makes use of the ratios of color channels according to the formula:

$$(a \leq (R/G) \leq b)\ \&\ (c \leq ((G+B)/R) \leq d)\ \&\ (e \leq (B/R) \leq f)$$

Such that *a*, *b*, *c*, *d*, *e*, and *f* are threshold values. This approach was presented by Omanovic, Buza and Besic, (Omanovic, 2014). See example output in Figure 14.

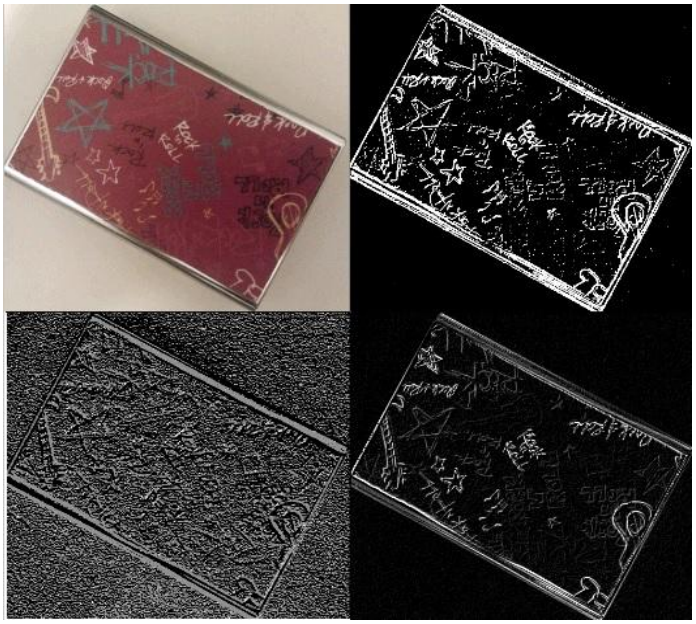

Figure 13:Original image, thresholded Sobel binary, gradient orientations, gradient magnitudes

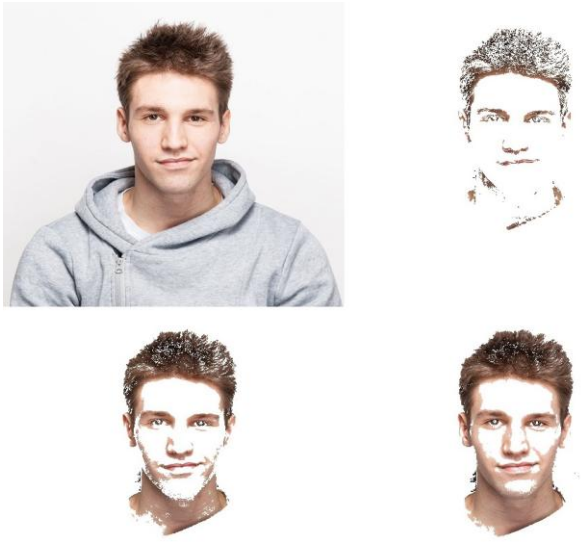

Figure 1: Output of the implemented skin color classificator, with decreasingly restrictive threshold parameters

3.3 Scale-Invariant Feature Transform

3.3.1 The Scale Space

The concept of a scale space filtering was first proposed by Andrew Witkin (Witkin, 1983). His method proposed a cascade filtering approach to represent and track signal extrema on multiple scales. Within computer vision the scale space structure is a widely used representation of visual information.

In cases where measurements (in our case images) are taken arbitrarily, it is impossible to estimate at which scale the objects or features of interest are to be found. If elements in an image are inspected at the scale of a tree, individual leaves can be considered insignificant. For the purpose of object recognition, however, the individual leaves or features at the corresponding scale, may be just as interesting as the entire tree itself. This is the motivation behind multi-scale representation of visual data in many computer vision algorithms.

In order to inspect data at differing scales it must be processed by a low pass filter with varying width. The Gaussian kernel function is proven to be the only possible function for this purpose. (Lowe, 2004)

The scale space is defined as

$$L(x, y; \sigma) = G(x, y; \sigma) * I(x, y)$$

where

$$L(x, y, \sigma)$$

Is the scale space function,

$$G(x, y, \sigma)$$

is the 2 Dimensional Gaussian function

$$I(x, y)$$

is the input image and

$*$

is the convolution operand.

The scale space structure consists of intervals and octaves, noted here as i and o. For each interval, i, the input image is low pass filtered with the Gaussian function with a scale parameter that is multiplied by a constant factor k, such that

$$\sigma_{oi} = \sigma_{oi-1} * k$$

and

$$\sigma_{00}$$

is implementation specific.

Experiments by Lowe settled for an initial sigma value of 1.6, which was determined to provide a good tradeoff between repeatability and efficiency. Use of incremental sigmas, as opposed to pre-compute the actual values, keeps the Gaussian window small, which decreases filtering time. For each octave the last image in the previous octave is down-scaled to half of the images dimensions. The resulting stack structure is called a Gaussian scale space pyramid, see figure 14.

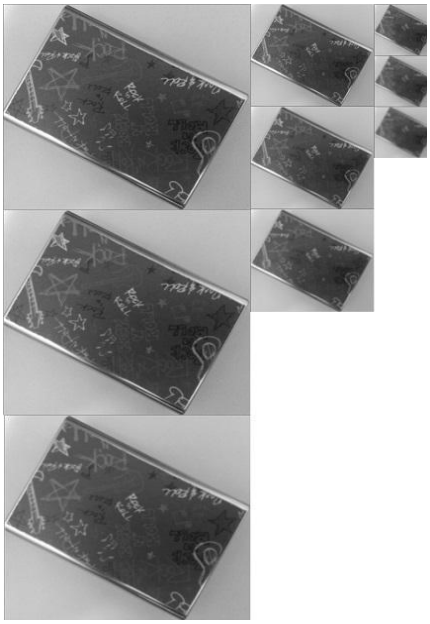

Figure 14: First three Octaves (columns) and intervals (Rows) of the Gaussian scale space.

3.3.2 Scale Space Extrema Localization

The difference of Gaussian function, abbreviated here DoG, is an approximation of the laplacian of Gaussian, but is significantly faster to compute, and is defined:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

In order to create a difference of Gaussian representation, neighboring images in the octave stack of the scale space are image-subtracted to produce a new stack of *n-1* DoG-images (where *n* is the amount of images in the scale space octave). Our interests lie at the local extrema of the scale space difference of Gaussian representation. A local extrema is the highest or lowest point of a function within a certain area. According to Fermat's theorem, local extrema of a function are found at points where the local derivatives are zero. In our case, local extrema are found by scanning the input DoG image for the highest or lowest values in the three dimensional difference of Gaussian neighborhood. The DoG neighborhood consists of the 26 neighboring pixels of the sample point, meaning the 3x3 pixels in the scales above and below, and the 3x3 pixels in the current scale, excluding the sample point (Lowe 2004), see Figure 15.

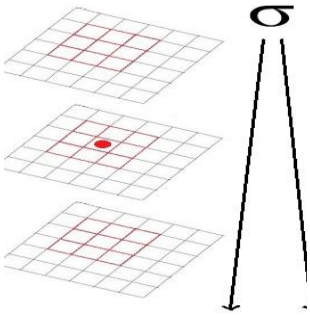

Figure 15: DoG neighborhood, with the sample pixel marked as a red dot

If a sampled value of *I(x, y, sigma)* is confirmed to be a local extrema, its approximate sub pixel location is estimated with the Taylor expansion (Lowe 2004) in order to improve the algorithms performance. Lowes first implementation did not calculate sub pixel locations, nor has the implementation presented in this chapter. Sub pixel calculation was found by Lowe to improve matching stability significantly (Lowe 2004).

3.3.3 Discarding Edges

In mathematics, a gradient describes the slope of a multidimensional surface, much like the derivative does for a two dimensional function. Since the difference of Gaussian responds well to edges, extrema are prone to be located at edges. In order to determine whether a keypoint is located at an edge, two perpendicular gradients are calculated at the keypoints location. The ratio of the gradients can determine whether a keypoint is located along an edge or flat region, this makes this approach well suited for discarding unwanted keypoints (Lowe 2004).

The ratio of the gradients can result in three different scenarios; both gradient will be small, meaning the region around the keypoint is 'flat', one gradient is large compared to the other, meaning the region is an edge or lastly, both gradients are large. In the last case, this would indicate a corner. The principal curvature describes how a surface bends at a point. It can mathematically be described with the Hessian matrix:

$$\begin{bmatrix} Dxx & Dxy \\ Dxy & Dyy \end{bmatrix}$$

The second order derivatives for the Hessian matrix are calculated by central approximation around the keypoint:

$$dxx = \frac{I(x+h, y) - 2I(x, y) + I(x-h, y)}{h^2}$$

The ratio of the eigenvalues of the Hessian matrix are proportional to the principal curvatures around the keypoint. The sum of the eigenvalues can be computed from the trace of the Hessian matrix, and their product from the determinant. By defining a threshold value, noted here *t*, the keypoint rejection can be done by checking against:

$$\frac{tr(H)^2}{|H|} < \frac{(t+1)^2}{t}$$

If the curvature ratio is larger than the threshold value, it is discarded as being too edge-like (Lowe 2004). See Figure 12 for output examples of the implemented difference of Gaussian keypoint detector.

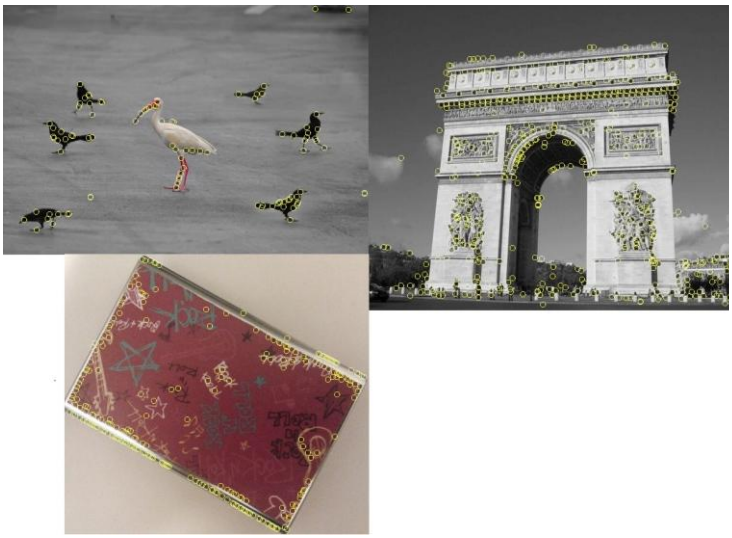

Figure 16: Output examples of the implemented difference of Gaussian keypoint detector (Unknown, Image1)

3.3.4 Assignment of Keypoint Orientation

Elements in an image may rotate for any reason. To achieve robustness against rotation it is necessary to be able to do comparisons of any kind relative to the amount of the experienced rotation. To achieve this, each keypoint is assigned one or several local orientations, so that any future calculations or comparisons can be done in relation to that specific orientation. To be able to assign an orientation to a keypoint, it is necessary to look for the greatest orientations in a region surrounding the keypoint. The gradient orientations and magnitudes for each image in the Gaussian scale space are pre-calculated and stored in memory for efficiency reasons. The implementation also allows for on-the-fly calculation of orientations and magnitudes. The resulting structures are, like the Gaussian scale space, a set of octaves and intervals containing the orientation an magnitude data.

The gradient magnitude and orientation for sample point $L(x, y)$ is defined:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\Theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

For each sampled orientation around the keypoint, the calculated magnitude at the sampled orientations location is added to an orientation histogram after has been weighed by a Gaussian window that is 1.5 times the sigma value for the current keypoint. The histogram consists of 36 bins, each bin covering 10 degrees of a full 360 degrees of orientation. A peak in the histogram indicates a dominant orientation in the sampled regions gradients.

For each peak within 80% of the highest peak a new keypoint is stored, and assigned the corresponding orientation. For keypoints with multiple dominant orientations this means that the location is stored several times but with different orientations. According to Lowe's experiments this logic contributes to improved matching stability. In order to get an even more accurate position for the histogram peak, a parabola is fitted around the histogram peak and its two neighboring bins to interpolate a more exact peak position (Lowe 2004).

3.3.5 Calculating the SIFT Feature Descriptor

As discussed in Chapter 2.4.2, the feature descriptor needs to be able to distinguish image features with high reliability, yet be robust enough to allow changes in as many factors as possible. The SIFT descriptor consists of a two-dimensional array of orientation histograms calculated around the location of the keypoint. The dimensionality of the SIFT descriptor experimentally determined by Lowe settled for a 4x4 descriptor of 8-binned histograms, resulting in a total of 128 values. The sample region was determined to be 16x16 pixels, resulting in 16 regions of dimensions 4x4. Each magnitude sample added to the histograms is weighed by its corresponding value in a Gaussian function with a sigma half of the descriptors width. The purpose of the Gaussian weighing window is to allow small changes in the descriptor window's position without causing significant changes in the descriptor histograms, and to give more significance to the values closer to the keypoints location. Once all 128 values have been computed, they are normalized in order to achieve tolerance to

illumination changes, since otherwise, a constant increase in contrast would affect the gradients. In order to achieve tolerance to changes in nonlinear illumination, the vector is also thresholded to truncate all values to a maximum of 0.2, a value experimentally determined by Lowe, and lastly re-normalized. The resulting 128 values constitute the feature vector, used for feature matching (Lowe 2004). See a visualization of the SIFT descriptor and feature vector in Figure 17 and tests performed with the implemented SIFT descriptor in figures 18 and 19.

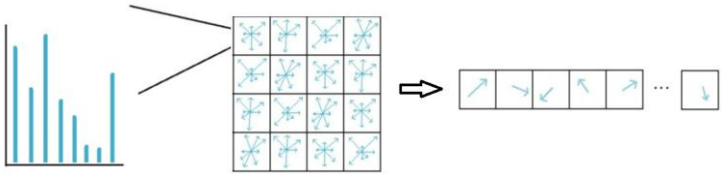

Figure 17: The SIFT descriptor (in the middle) consists of 4x4 histograms generated from 16 image regions surrounding a keypoint. From the histogram values a feature vector of 128 values (on the right) is generated. Arrows indicate the histogram bins' orientation, and their length the bins' magnitude.

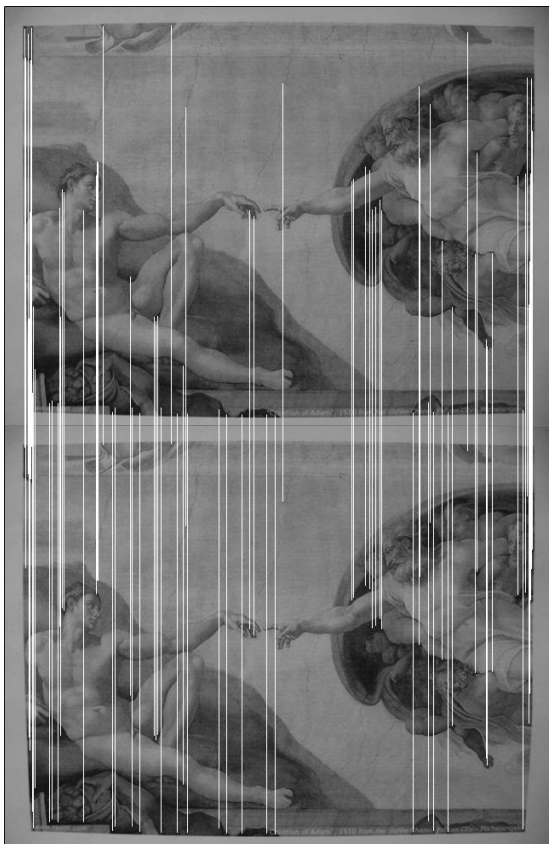

Figure 18: Feature matching without change in viewpoint (original image: Yu, 2011)

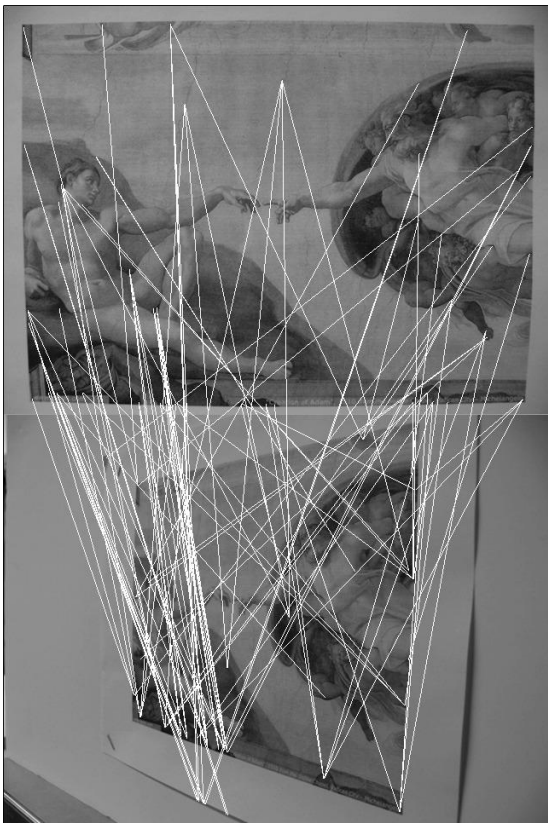

Figure 19: Feature matching with significant change in viewpoint. The simplest criteria possible, thresholded Euclidean distance, was used as a matching criteria. Note the many false positives. (original image: Yu, 2011)

4 LIBRARY STRUCTURE

Within the scope of this thesis an image processing and computer vision function library was implemented for the purpose of independent computer vision. By keeping the library pure of external dependencies we eliminate the risk of restrictions such libraries and code may bring with them to the project. By using own data structures and limiting the use of system functions we minimize, and perhaps even eliminate, the workload when porting code to new platforms.

At the time of writing the library supports some (but not all) of the most important basic features such a library needs to have, including but not limited to, file I/O, file format conversion, image scaling/pre-processing and feature-based algorithms. The algorithms have been standardized to use double precision floating point images, converted from the PPM file format as the standard input type of the functions. In order to keep the implementation logic as simple as possible, some implementations have been separated for example for gray scaled and full-color images. These choices are necessary since image processing algorithms should optimally contain as little logic and decision making as possible. Therefore, wrapper functions have been implemented to apply the appropriate functions based on simple heuristics. Some examples include automatic choice of image scaling algorithm, where the images destination dimensions are evaluated to pick a scaling algorithm from either nearest neighbor (better suited for down scaling) or bi-linear interpolation (better suited for up scaling), or the automatic picking of filtering algorithm depending on the amount of color channels the image contains.

Some other design choices are the support of callback functions for image arithmetic. This callback functionality allows the programmer to implement a decision-making function which affects how the operations affect the end result, such as value thresholding or similar inspection of values prior outputting them. This type of processing can be useful for example in segmentation algorithms that include a step of background subtraction. While this is unlikely to affect execution time, it may save the trouble of creating a separate step in the image processing pipeline. If the raw results are of interest, the callback processing can be omitted. It was chosen that the image processing algorithms would only support floating point

images, this way it is only necessary to implement conversion from other formats, as opposed to re-implement the functions all over to support new image formats.

The library is implemented in the C programming language and is tested in the Linux environment. The programming language C was chosen for the implementations, as it is proven to be highly efficient performance-wise and allows flexible use of system resources (Attractivechaos, 2011 . bioinformatics.org). The compiler used during testing is the gcc GNU C-compiler, and the gcc preprocessor for macro functions. The implemented function library is divided into several files and their corresponding file headers. Each file contains functions for a specific purpose.

binppmio.c / binppmio.h:

Functions specific for the PPM file format.

- Image input/output for the binary PPM format.
- Basic image operations like memory allocation and de-allocation, operations with the image format file-headers and adjusting of color information

floatimgio.c / floatimgio.h:

Functions specific for the floating point image format.

- Structural conversions from, and to, the double precision floating point image format.
- Basic image operations like memory allocation and de-allocation, operations with the image format file-headers and adjusting of color information

filter.c / filter.h:

Functions central for the convolution filters.

- convolution filtering
- built in filters

- generation of the Gaussian filter.

floatcvops.c / floatcvops.h:

General operations that do not fit in other libraries.

- image scaling
- image arithmetics
- skincolor detection

fsift.c / fsift.h:

Functions related to the scale invariant feature transform.

- difference of Gaussian feature detector
- SIFT feature descriptor
- keypoint matching algorithms

In addition some collections of supporting functions and data structures.

datastructures.c / datastructures.h:

data structures for storing keypoint and descriptor data

nmath.c / nmath.h:

collection of supporting math functions. For example matrix operations, distance functions and image gradients.

neko.h:

collection of constants, macro functions and included standard libraries used in the files above.

4.1.1 Performance Evaluation

In many computer vision applications it is desirable to have the results in real-time. Therefore it is in many cases imperative that processing algorithms are fast and efficient. Image pre-processing time is therefore a suitable metric for benchmarking library speed performance, as it is a central part of many algorithms and can potentially cause application-breaking slowness. It is also as a step in the image pre-processing pipeline, that is shared by many algorithms. It was chosen to use the pre-processing algorithms presented in Chapter 3 as a metric for the purpose of evaluating the speed performance of the library. Convolution filters of varying sizes were applied to images of varying sizes to collect a general idea of how filter and image sizes and image color models affect the processing time. Each combination was calculated 50 times and the average filtering times for each combination are presented in the Figures 14 and 15, plotted image filtering time in milliseconds (y-axis) vs image and filter size in pixels (x-axis). See appendix spreadsheet of filtering times and estimated video frame rates.

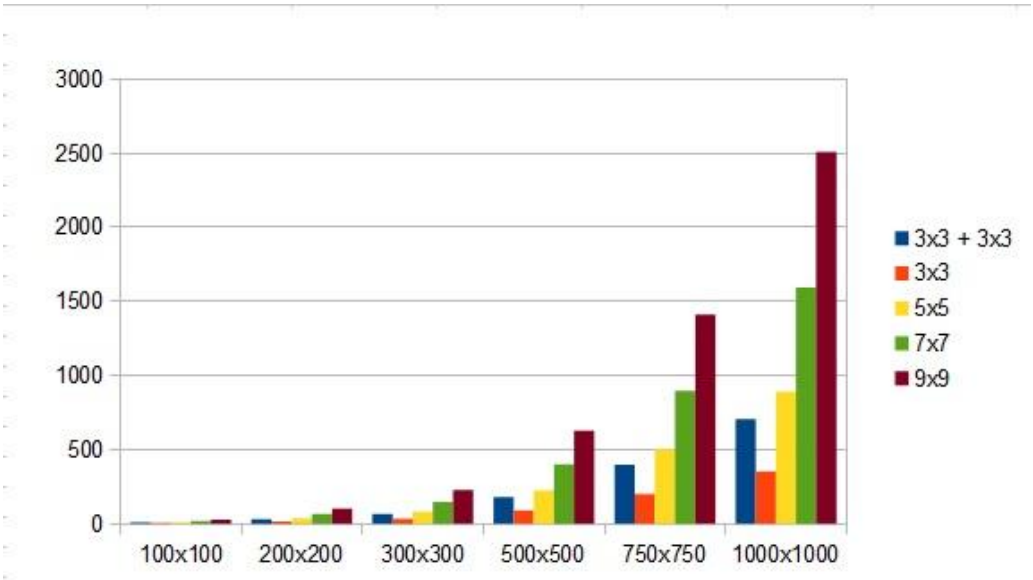

Figure 20: Average filtering times for gray scale images

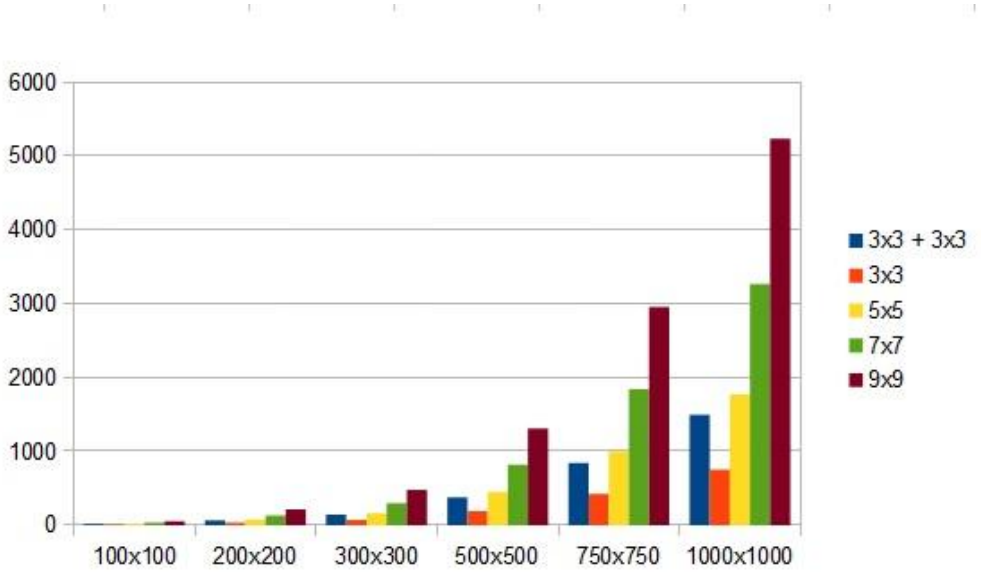

Figure 21: Average filtering times for RGB images

From the filtering times we can approximate how given filter and image sizes affect the frame rate of video files of same sizes. A theoretical frame rate of 25 fps, where each frame is displayed for 40 milliseconds, is used as reference, see approximated frame rates plotted in figures 16 and 17.

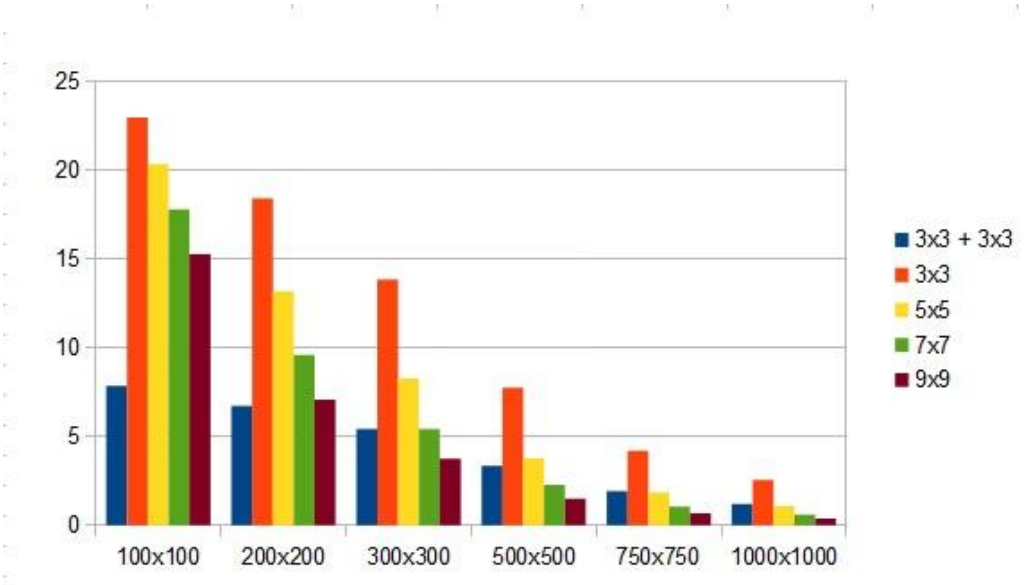

Figure 22: Estimated affected frame rates for filtered gray scale video of 25 fps

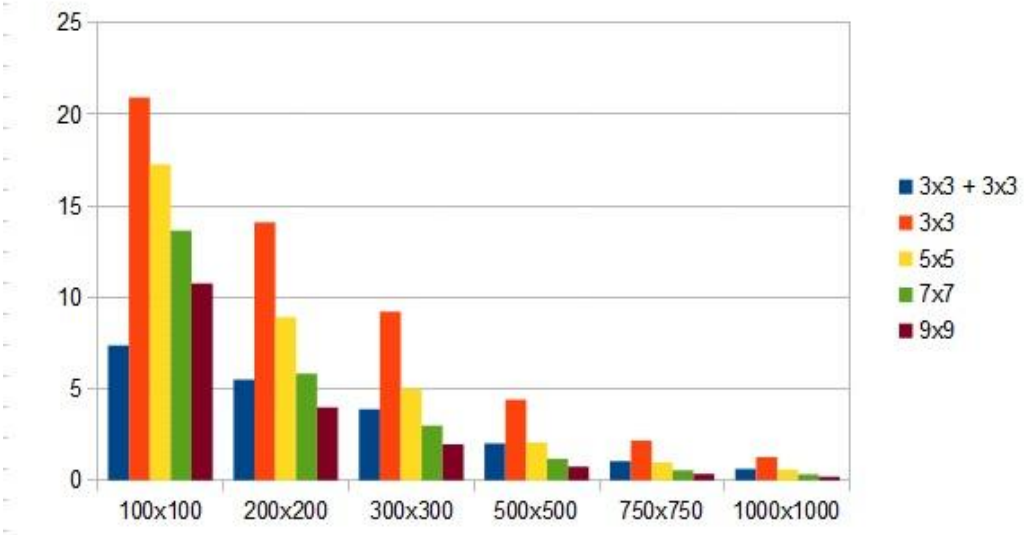

Figure 23: Estimated affected frame rates for filtered RGB video of 25 fps

The results presented were calculated with an Intel i686 -architecture 3Ghz CPU from a set of test images. The results show, as expected, that filtering times are significantly smaller for

gray scale images as opposed to RGB images and that increasing image and filter sizes also affects execution time significantly. The linearity of the time complexity can also be observed.

5 CONCLUSIONS AND FUTURE WORK

This thesis evaluated algorithms for image processing and computer vision, for the purpose of creating a new function library for independent implementations of said algorithms. The algorithms implemented in the presented library provide some essential techniques for image processing and computer vision. The functionality and performance of the presented algorithms has been deemed largely satisfactory and the library is concluded to provide an excellent foundation to be extended upon.

As can be seen from the figures in the performance evaluation done in chapter 4, the processing times become too long for real-time filtering for video of high resolution. In order to achieve faster processing times the filtering algorithms should be implemented to support separable filters. This would significantly increase filtering time. Adding support for parallellization of the processing algorithms would also provide additional performance boosts.

The performance of the scale invariant feature transform was during testing deemed unsatisfactory for the purpose of reliable feature matching, further development is necessary in order to improve the detectors repeatability, and more robust matching strategies need to be evaluated. By adding the currently missing sub pixel interpolation, the algorithms robustness can be improved, as determined by Lowe. Additional detectors, such as the Harris corner detector and the Canny edge detector should be implemented to evaluate whether they are suitable for robust feature detection.

Currently missing critical features are support for geometric transformations and segmentation. The development process is still ongoing and these features will be added in the future.

6 REFERENCES

Agoston, Max. 2005, *Computer Graphics and Geometric Modeling.* Springer ISBN 978-1-85233-818-3.

Attractivechaos.github.io. 2011, *Benchmark language implementations*. Available at: https://attractivechaos.github.io/plb/ . Accessed 17.5.2015.

Bioinformatics.org, Language Benchmark. Available at: http://www.bioinformatics.org/benchmark/results.html. Accessed 17.5.2015.

Chang, Jeannette; Arbelaez, Pablo; Switz Neil; Reber, Clay; Tapley, Asa ; Davis, Lucian; Cattamanchi, Adithya; Fletcher, Daniel; Malik, Jitendra. 2012, *Automated Tuberculosis Diagnosis Using Fluorescence Images from a Mobile Microscope*, UC Berkeley Department of Electrical Engineering and Computer Sciences, UC Berkeley Department of Bioengineering, UC San Francisco Medical School and San Francisco General Hospital.

Chung, Moo K., The Gaussian Kernel, Available at http://www.stat.wisc.edu/~mchung/teaching/MIA/reading/diffusion.gaussian.kernel.pdf.pdf. Accessed 20.5.2015

Hays, 2013a, *image formation and filtering*, Available at: http://cs.brown.edu/courses/cs143/lectures/03.pdf. Accessed 20.5.2015.

Hays, 2013b James, *Interest Points and Corners*, Available at: http://cs.brown.edu/courses/cs143/lectures/08.pdf. Accessed 20.5.2015.

Hays, 2013c *Edge detection*, Available at: http://cs.brown.edu/courses/cs143/lectures/07.pdf. Accessed 20.5.2015.

Hess, Rob. 2010, *OpenSIFT, an Open-Sourse SIFT Library*, Available at http://robwhess.github.io/opensift/siftlib-acmmm10.pdf. Accessed 17.5.2015.

Lowe, David. 2005, *Distinctive Image Features from Scale-Invariant Keypoints* Computer Science Department, University of British Columbia, Canada: Vancouver.

Mettler-Toledo. 2013, *Measuring With Vision Inspection In Plastic Bottle Manufacturing*, USA, Illinois: Aurora.

Mikolajczyk, Krystian ; Schmid, Cordelia. 2005, *A Performance Evaluation of Local Descriptors.*

Munroe, Randall. *XKCD: Tasks*. Available at: http://xkcd.com/1425/. Accessed 17.5.2015.

Myler, Harley R; Weeks, Arthur R. *The Pocket Handbook of Image Processing Algorithms in C*. ISBN 0-13-642240-3

NetPBM, 2013, *PPM Format Specification*. Available at: http://netpbm.sourceforge.net/doc/ppm.html. Accessed 20.5.2015.

OpenCV OpenCV, Available at: http://opencv.org/. Accsessed 20.5.2015

Samir, Omanovic; Besic, Ingmar; Buza, Emir;. 2014, *RGB ratios based skin detection*, Department for Computer Science and Informatics Faculty of Electrical Engineering, Bosnia and Herzegovina: Sarajevo.

Schmid, Cordelia; Mohr, Roger; Bauckhage, Christian. 2000, *Evaluation of Interest Point Detectors*. France: Montbonnot.

Schubert E. Fred. 2006, *"Human Eye Sensitivity and Photometric Quantities," in Light Emitting Diodes*. Cambridge University Press.

Szeliski, Richard. 2011, *Computer Vision: Algorithms and Applications*. Springer. ISBN 978-1-84882-934-3

VLFeat *VLFeat*, Available at http://www.vlfeat.org/. Accessed 17.5.2015.

Witkin, Andrew P. 1983, *Scale-Space Filtering*, Fairchild Laboratory for Artificial Intelligence Research.

Yu, Guoshen. 2011, *SIFT and ASIFT*, Available at: http://www.cmap.polytechnique.fr/~yu/research/ASIFT/. Accessed 20.5.2015.

Unknown, *Image1*, Available at https://www.flickr.com/photos/drurydrama/. Accessed 20.5.2015.

Unknown, Image2, Available at http://commons.wikimedia.org/wiki/User:SharkD. Accessed 20.5.2015.

BILAGOR / APPENDICES

– Swedish summary

Appendix 1

Sammandrag

Erik Granholm

Datorseende är ett område inom datavetenskap som innebär att extrahera underliggande information ur bilddata, med avsikt att kunna agera utgående från bildens innehåll. Människan kan lätt tolka färg- och rymdinformation, särskilja för- och bakgrund ifrån varandra samt följa och klassificera objekt och personer från video. Även med ständigt ökande beräkningskapacitet har datorer svårigheter att nå nivån av ett två år gammalt barn. Detta examensarbete utvärderar bildbehandlings algoritmer, samt algoritmer för upptäckning, beskrivning samt jämförelse av såkallade bildkännetecken. Dylika algoritmer har även implementerats i ett funktionsbibliotek för självständig och resurssnål datorseende, som också beskrivs i detta examensarbete.

Bildkännetecken är små områden i en bild, vilka kan sägas innehålla intressant information. De används inom datoreseende för att exempelvis söka objekt i bilder. En typisk algoritm är tredelad, kännetecknen upptäcks med en detektor algoritm, varefter de omvandlas till en så kallad kännetecken deskriptor, vilket beskriver området i bilden runt kännetecknet. Till slut Används deskriptorerna för att söka fram liknande kännetecken från bilder eller video för att uppskatta om kännetecknen är lika. Det här examensarbetet beskriver det implementerade SIFT (Scale Invariant Feature Transform) detektorn samt deskriptorn för bildkännetecken, samt bildbehandlingsprocesser och algoritmer.

Det implementerade biblioteket har skrivits i programmeringsspråket C, och består av flera källkodsfiler med korresponderande header filer. Biblioteket har testats i en Linux omgivning och den slutsatsen dragits, att funktionsbiblioteket presterar väl och innehåller de viktigate grundfunktionaliteten för datorseende. För att få prestandan på toppnivå måste bilbioteket ytterligare utvecklas, samt en del funktionalitet tillsättas för att uppnå all viktigaste grundfunktionalitet som behövs för datorseende av generellt syfte. Utvecklingen pågår ännu och funktionalitet kommer att tillsättas under projektets gång.